UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/705,525	11/10/2003	Attila Barta	RSW920030176US1	5400

51016 7590 10/15/2008

IBM CORP. (RALEIGH SOFTWARE GROUP)
c/o Rudolf O Siegesmund Yee & Associates, P.C.
P.O. Box 802333
DALLAS, TX 75380

EXAMINER	
CHEN, QING	

ART UNIT	PAPER NUMBER
2191	

MAIL DATE	DELIVERY MODE
10/15/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/705,525	BARTA ET AL.	
	Examiner	Art Unit	
	Qing Chen	2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *19 June 2008*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1,3-13 and 41* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1,3-13 and 41* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This Office action is in response to the amendment filed on June 19, 2008.

2. **Claims 1, 3-13, and 41** are pending.

3. **Claims 1, 9, 12, and 13** have been amended.

4. **Claims 2 and 14-40** have been cancelled.

5. **Claim 41** has been added.

6. The objections to Claims 1, 3-14, 16-20, and 34-40 are withdrawn in view of Applicant's amendments to the claims or cancellation of the claims.

7. The 35 U.S.C. § 112, first paragraph, rejections of Claims 1, 3-14, 16-19, 21, and 23-40 are withdrawn in view of Applicant's amendments to the claims or cancellation of the claims.

8. The 35 U.S.C. § 112, second paragraph, rejections of Claims 1, 3-13, 35, and 39 are withdrawn in view of Applicant's amendments to the claims or cancellation of the claims.

Response to Amendment

Claim Objections

9. **Claim 41** is objected to because of the following informalities:

- **Claim 41** contains a typographical error: "[A] third set of software components has dependencies only the first and second sets of software components" should read -- a third set of software components has dependencies only on the first and second sets of software components --.

- **Claim 41** recites the limitation "the method." Applicant is advised to change this limitation to read "the computer implemented method" for the purpose of providing it with proper explicit antecedent basis.

Appropriate correction is required.

Claim Rejections - 35 USC § 103

10. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

11. **Claims 1, 3, 4, and 6-13** are rejected under 35 U.S.C. 103(a) as being unpatentable over **US 6,442,754 (hereinafter "Curtis")** in view of **US 6,725,452 (hereinafter "Te'eni")** and **US 6,675,382 (hereinafter "Foster")**.

As per **Claim 1**, Curtis discloses:

- using a data structure in a storage that provides, for each of the plurality of software components from the application, a software component deployment dependency data, an indication of necessary software components for an operation of each of the plurality of software components being installed *(see Figure 5; Column 13: 7-27 and 33-37, "... a data structure ... is maintained in the registry object or registry database 220, indicating installed programs and dependent components on which installed programs depend. In the embodiment of FIG. 5, the*

data structure is a hierarchical arrangement of programs, file sets, and dependent components in the form of a directory tree." and "Each installed file set component has a Dependency subdirectory which includes information on each dependent component on which the file set and program depend in order to operate. The dependency subdirectory would list the program name, version, fileset name, and fileset version for each program on which the fileset including the dependency subdirectory depends." and "... the dependency directory may indicate dependent file sets or registry objects that are the subject matter of the processed dependency object. If there are no dependent components, then the dependency directory will contain no values."); and

- using a computer connected to the storage and a program installed in a memory of the computer *(see Figure 1: 10; Column 5: 29-31, "The programs in memory 12 includes an operating system (OS) 16 program and application programs, such as an install program 17 or an installer tool kit.")*, performing the steps of:

- determining a first plurality of software components previously installed on a system *(see Column 11: 11-20, "... a call to the check_dependency function ... This function determines whether the file, program or registry object indicated in the dependency object 400 is installed on the computer.")*;

- determining a second plurality of software components to be installed on the system *(see Figure 2: 340; Column 11: 23-24, "... a file set 340 is installed.")*;

- accessing a third plurality of software component deployment dependency data *(see Column 13: 18-21, "Each installed file set component has a Dependency subdirectory*

which includes information on each dependent component on which the file set and program depend in order to operate."); and

- accessing a sixth plurality of metadata from the data structure regarding the second plurality of software components to be installed and accessing a seventh plurality of metadata regarding the first plurality of software components previously installed *(see Column 13: 13-15 and 21-24, "A root directory includes a sub-directory for each installed program, indicating the program name and version." and "The dependency subdirectory would list the program name, version, fileset name, and fileset version for each program on which the fileset including the dependency subdirectory depends.").*

However, Curtis does not disclose:

- an indication of incompatibility with a previously installed software component;
- determining a fourth plurality of software components suitable for parallel installation;
- determining an order in which the fourth plurality of software components can be grouped for a fifth plurality of parallel installations;
- analyzing the sixth plurality of metadata to determine an eight plurality of potential conflicts between the second plurality of software components to be installed and the first plurality of software components previously installed on the system;
- wherein a pre-deployment analysis allows the second plurality of software components to be installed in parallel and in a sequence of groups; and
- wherein an installation time for the application is reduced.

Te'eni discloses:

- an indication of incompatibility with a previously installed software component *(see Column 1: 61-64, "When performing the predefined procedures necessary for an upgrade to be implemented frequently dependency conflicts may arise among the components present and the components to be installed."; Column 5: 10-17, "Virtual upgrade module 36 creates upgrade processes for the following tasks that are performed sequentially: (a) to collect all the information necessary for the dependency analysis and the dependency conflicts resolving process such as the relevant component data information units from component data table 28, the encoded dependency rules from xor-rules table 32 and from add-remove rules table 34 module ...")*; and

- analyzing the sixth plurality of metadata to determine an eight plurality of potential conflicts between the second plurality of software components to be installed and the first plurality of software components previously installed on the system *(see Column 1: 61-64, "When performing the predefined procedures necessary for an upgrade to be implemented frequently dependency conflicts may arise among the components present and the components to be installed."; Column 5: 10-20, "Virtual upgrade module 36 creates upgrade processes for the following tasks that are performed sequentially: (a) to collect all the information necessary for the dependency analysis and the dependency conflicts resolving process such as the relevant component data information units from component data table 28, the encoded dependency rules from xor-rules table 32 and from add-remove rules table 34 module, (b) to activate conflict resolver module 40 in order to check for potential dependency conflicts and to resolve any dependency conflicts that might arise as a result of the planned installation process ...")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Te'eni into the teaching of Curtis to include an indication of incompatibility with a previously installed software component; and analyzing the sixth plurality of metadata to determine an eight plurality of potential conflicts between the second plurality of software components to be installed and the first plurality of software components previously installed on the system. The modification would be obvious because one of ordinary skill in the art would be motivated to prevent any unsuccessful installation *(see Te'eni – Column 1: 64-66)*.

Foster discloses:

- determining a fourth plurality of software components suitable for parallel installation *(see Column 10: 6-8, "... other packages may be concurrently installed that require the presence of package 200 on the system.")*;

- determining an order in which the fourth plurality of software components can be grouped for a fifth plurality of parallel installations *(see Column 10: 8-10, "... the system checks the dependencies between package 200 and the packages that are being simultaneously installed.")*;

- wherein a pre-deployment analysis allows the second plurality of software components to be installed in parallel and in a sequence of groups *(see Column 10: 8-10, "... the packages that are being simultaneously installed.")*; and

- wherein an installation time for the application is reduced *(see Column 10: 8-10, "... the packages that are being simultaneously installed.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Foster into the teaching of Curtis to include determining a fourth plurality of software components suitable for parallel installation; determining an order in which the fourth plurality of software components can be grouped for a fifth plurality of parallel installations; wherein a pre-deployment analysis allows the second plurality of software components to be installed in parallel and in a sequence of groups; and wherein an installation time for the application is reduced. The modification would be obvious because one of ordinary skill in the art would be motivated to provide an efficient and simple solution for packaging, distributing and installing software *(see Foster – Column 1: 43-44)*.

As per **Claim 3**, the rejection of **Claim 1** is incorporated; and Curtis further discloses:

- updating the data structure with an identity of a ninth plurality of recently installed software components *(see Column 13: 28-30, "The information in this directory is created whenever a component is installed. For instance, whenever a program is installed, a subdirectory is created under the root directory.")*.

As per **Claim 4**, the rejection of **Claim 1** is incorporated; and Curtis further discloses:

- providing a user with a plurality of options for the eight plurality of potential conflicts *(see Column 12: 35-45, "If so, control transfers to block 534 where the program displays on the display means 14 the name of the dependent component that was not located on the system and a radio button to allow the user to selectively cause the execution of the install script in the Install 416 or SInstall 418 fields. If there is not install script, then control transfers to block 536 where*

the program displays information maintained in the install information field 426 to inform the user on where to obtain the dependent component that is needed before the program may be installed.").

As per **Claim 6**, the rejection of **Claim 4** is incorporated; and Curtis further discloses:

- wherein a second option includes continuing an installation *(see Column 12: 51-53, "When the user selects to install the file, the install program 17 will execute the install script in either the Install 416 or SInstall field 418.").*

As per **Claim 7**, the rejection of **Claim 6** is incorporated; and Curtis further discloses:

- upon the exercise of the second option, recording an entry in a log indicative of a conflict and of a continuation of installation *(see Figure 2: 140; Column 7: 4-5, "During install, the log 140 and 'uninstall.Java1' 150 information are built."; Column 8: 24-29, "... providing various logs, e.g. a log for keeping track of what is being installed, and a log that reports the progress of install. Logs are used for both the install and uninstall process. Furthermore, these logs are human readable which allows them to be checked, e.g., after a silent install, to ensure that a file has installed successfully.").*

As per **Claim 8**, the rejection of **Claim 1** is incorporated; and Curtis further discloses:

- initiating a removal of a software component from a system *(see Figure 6: 560; Column 13: 50-51, "... the program processes a request to uninstall a program.")*; and

- identifying a tenth plurality of remaining software components which depend on the software component to be removed *(see Column 13: 59-62, "The uninstall program may navigate the directory structure from the dependency directory shown in FIG. 5 to determine dependant programs that depend on the program subject to the uninstallation.").*

As per **Claim 9**, the rejection of **Claim 8** is incorporated; and Curtis further discloses:

- providing a user with a plurality of options if the tenth plurality of dependent remaining software components are identified *(see Column 12: 35-45, "If so, control transfers to block 534 where the program displays on the display means 14 the name of the dependent component that was not located on the system and a radio button to allow the user to selectively cause the execution of the install script in the Install 416 or SInstall 418 fields. If there is not install script, then control transfers to block 536 where the program displays information maintained in the install information field 426 to inform the user on where to obtain the dependent component that is needed before the program may be installed.").*

As per **Claim 10**, the rejection of **Claim 9** is incorporated; and Curtis further discloses:

- wherein a first option includes aborting a removal *(see Figure 6: 570; Column 13: 62-63, "Control then transfers to block 570 to exit uninstallation ...").*

As per **Claim 11**, the rejection of **Claim 9** is incorporated; and Curtis further discloses:

- wherein a second option includes continuing a removal *(see Figure 6: 568; Column 13: 55-56, "Otherwise, control transfers to block 568 to proceed with the uninstallation.").*

As per **Claim 12**, the rejection of **Claim 8** is incorporated; and Curtis further discloses:

- identifying a first software component previously installed on a system which is dependent upon a removed software component *(see Column 13: 4-6 and 64-67, "... before uninstalling a program, a determination may be made as to whether other installed components depend on the file being uninstalled." and "... information indicating the depending programs that should be uninstalled before continuing with the uninstallation of the program, which is a dependent program.")*; and

- determining an identity of a second software component upon which the first software component depends *(see Column 13: 1-4, "During installation of a dependent program, dependency information from the Dependency Object 400 may be written to a dependency location indicating dependent components of the installed file.")*.

As per **Claim 13**, the rejection of **Claim 12** is incorporated; and Curtis further discloses:

- installing the second software component upon which the first software component depends *(see Column 13: 1-4, "During installation of a dependent program, dependency information from the Dependency Object 400 may be written to a dependency location indicating dependent components of the installed file.")*; and

- creating a dependency link between the first software component and the second software component *(see Column 13: 1-4, "Dependency Object 400 may be written to a dependency location indicating dependent components of the installed file.")*.

12. **Claim 41** is rejected under 35 U.S.C. 103(a) as being unpatentable over **Curtis** in view of **Foster**.

As per **Claim 41**, Curtis discloses:

- accessing the semantic model to obtain a dependency information about the application software components *(see Figure 5; Column 13: 7-27 and 33-37, "... a data structure ... is maintained in the registry object or registry database 220, indicating installed programs and dependent components on which installed programs depend. In the embodiment of FIG. 5, the data structure is a hierarchical arrangement of programs, file sets, and dependent components in the form of a directory tree." and "Each installed file set component has a Dependency subdirectory which includes information on each dependent component on which the file set and program depend in order to operate. The dependency subdirectory would list the program name, version, fileset name, and fileset version for each program on which the fileset including the dependency subdirectory depends." and "... the dependency directory may indicate dependent file sets or registry objects that are the subject matter of the processed dependency object. If there are no dependent components, then the dependency directory will contain no values.")*; and

- using the dependency information to group the application software components into sets of software components with like dependency levels, wherein a first set of software components has no dependencies, a second set of software components has dependencies only on the first set of software components, and a third set of software components has dependencies only on the first and second sets of software components *(see Figure 5; Column 13: 7-27 and*

33-37, "... a data structure ... is maintained in the registry object or registry database 220, indicating installed programs and dependent components on which installed programs depend. In the embodiment of FIG. 5, the data structure is a hierarchical arrangement of programs, file sets, and dependent components in the form of a directory tree." and "Each installed file set component has a Dependency subdirectory which includes information on each dependent component on which the file set and program depend in order to operate. The dependency subdirectory would list the program name, version, fileset name, and fileset version for each program on which the fileset including the dependency subdirectory depends." and "... the dependency directory may indicate dependent file sets or registry objects that are the subject matter of the processed dependency object. If there are no dependent components, then the dependency directory will contain no values.").

However, Curtis does not disclose:

- installing the first set of software components in parallel;
- responsive to completing installation of the first set of software components, installing the second set of software components in parallel; and
- responsive to completing installation of the second set of software components, installing the third set of software components in parallel.

Foster discloses:

- installing the first set of software components in parallel *(see Column 10: 6-10, "... other packages may be concurrently installed that require the presence of package 200 on the system." and "... the packages that are being simultaneously installed.")*;

- responsive to completing installation of the first set of software components, installing the second set of software components in parallel *(see Column 10: 6-10, "... other packages may be concurrently installed that require the presence of package 200 on the system." and "... the packages that are being simultaneously installed.")*; and

- responsive to completing installation of the second set of software components, installing the third set of software components in parallel *(see Column 10: 6-10, "... other packages may be concurrently installed that require the presence of package 200 on the system." and "... the packages that are being simultaneously installed.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Foster into the teaching of Curtis to include installing the first set of software components in parallel; responsive to completing installation of the first set of software components, installing the second set of software components in parallel; and responsive to completing installation of the second set of software components, installing the third set of software components in parallel. The modification would be obvious because one of ordinary skill in the art would be motivated to provide an efficient and simple solution for packaging, distributing and installing software *(see Foster – Column 1: 43-44).*

13. **Claim 5** is rejected under 35 U.S.C. 103(a) as being unpatentable over **Curtis** in view of **Te'eni** and **Foster** as applied to Claim 4 above, and further in view of **US 6,918,112 (hereinafter "Bourke-Dunphy")**.

As per **Claim 5**, the rejection of **Claim 4** is incorporated; however, Curtis, Te'eni, and Foster do not disclose:

- wherein a first option includes aborting an installation.

Bourke-Dunphy discloses:

- wherein a first option includes aborting an installation *(see Figure 5: 236; Column 8: 35-38, "... the user may select a CANCEL action button 236 to return to the component selection user interface ... where the user may manually modify the component selections.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Bourke-Dunphy into the teaching of Curtis to include wherein a first option includes aborting an installation. The modification would be obvious because one of ordinary skill in the art would be motivated to allow the user to exit the current installation, correct the error identified, and reinitiate the installation procedure *(see Bourke-Dunphy – Column 1: 27-34).*

Response to Arguments

14. Applicant's arguments filed on June 19, 2008 have been fully considered, but they are not persuasive.

In the Remarks, Applicant argues:

a) The present invention prevents these types of well known conflicts from occurring in the first place. "[A] pre-deployment analysis" is performed "prior to deployment of the application." During this analysis, a data structure is read to determine which (if any) software components to

be installed will conflict with previously installed software components. Because the data structure contains "an indication of incompatibility," the analysis allows the user to abort installation or remedy the incompatibility before deployment of the application occurs. Thus, claim 1 prevents conflicts or incompatibilities between software components from occurring. Te'eni 1:61-64, by contrast, simply teaches that conflicts can occur when a computer system is upgraded. Accordingly, Te'eni 1:61-64 fails to teach "a data structure that provides, for each of the plurality of software components, ... an indication of incompatibility with a previously installed software components." Teaching that conflicts are known to occur is not the same as providing a data structure that provides an indication of incompatibility prior to deployment of an application.

Examiner's response:

a) Examiner disagrees. Applicant's arguments are not persuasive for at least the following reasons:

First, in response to the Applicant's arguments against the references individually, one cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981); *In re Merck & Co.,* 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986).

Second, with respect to the Applicant's assertion that Te'eni fails to teach the claimed feature of "a data structure [...] that provides, for each of the plurality of software components, [...] an indication of incompatibility with a previously installed software component," as previously pointed out in the Non-Final Rejection (mailed on 03/19/2008) and further clarified

hereinafter, the Examiner respectfully submits that Te'eni is relied upon by the Examiner for its specific teaching of "an indication of incompatibility with a previously installed software component." Note that Curtis clearly discloses "using a data structure [...] that provides, for each of the plurality of software components [...], a software component deployment dependency data, an indication of necessary software components for an operation of each of the plurality of software components being installed" *(see Figure 5; Column 13: 7-27 and 33-37, "... a data structure ... is maintained in the registry object or registry database 220, indicating installed programs and dependent components on which installed programs depend. In the embodiment of FIG. 5, the data structure is a hierarchical arrangement of programs, file sets, and dependent components in the form of a directory tree." and "Each installed file set component has a Dependency subdirectory which includes information on each dependent component on which the file set and program depend in order to operate. The dependency subdirectory would list the program name, version, fileset name, and fileset version for each program on which the fileset including the dependency subdirectory depends." and "... the dependency directory may indicate dependent file sets or registry objects that are the subject matter of the processed dependency object. If there are no dependent components, then the dependency directory will contain no values.")*. However, Curtis does not disclose using a data structure that provides "an indication of incompatibility with a previously installed software component." Te'eni clearly discloses "an indication of incompatibility with a previously installed software component" *(see Column 1: 61-64, "When performing the predefined procedures necessary for an upgrade to be implemented frequently dependency conflicts may arise among the components present and the components to be installed."; Column 5: 10-17, "Virtual upgrade module 36 creates upgrade*

processes for the following tasks that are performed sequentially: (a) to collect all the information necessary for the dependency analysis and the dependency conflicts resolving process such as the relevant component data information units from component data table 28, the encoded dependency rules from xor-rules table 32 and from add-remove rules table 34 module ..."). Thus, in view of Te'eni, one of ordinary skill in the art would be motivated to include an indication of incompatibility with a previously installed software component in the data structure of Curtis in order to prevent any unsuccessful installation *(see Te'eni – Column 1: 64-66).*

Third, Examiner further submits that the process of storing an indication of incompatibility with a previously installed software component is well-known to one of ordinary skill in the computing art and also conventional in the area of software installation. By way of an example and not of limitation, software installation tools commonly include a software component dependency analyzer to collect and analyze software component dependency information and alert the user if a potential dependency conflict exists.

Therefore, for at least the reasons set forth above, the rejection made under 35 U.S.C. § 103(a) with respect to Claim 1 is proper and therefore, maintained.

In the Remarks, Applicant argues:

b) Similarly, the Examiner alleges that, although Curtis fails to teach it, Te'eni 1:61-64 teaches "analyzing the sixth plurality of metadata to determine an eighth plurality of potential conflicts." Office Action p. 17. However, as explained above, Te'eni 1:61-64 simply teaches that dependency conflicts may arise when a computer system is upgraded. This is not the same as

analyzing metadata to determine potential conflicts. Whereas claim 1 refers to a pre-deployment analysis used to prevent conflicts from occurring, Te'eni 1:61-64 refers to the fact that conflicts can occur when a computer is upgraded.

Examiner's response:

b) Examiner disagrees. Applicant's arguments are not persuasive for at least the following reasons:

First, in response to the Applicant's arguments against the references individually, one cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981); *In re Merck & Co.,* 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986).

Second, with respect to the Applicant's assertion that Te'eni fails to teach the claimed feature of analyzing metadata to determine potential conflicts, as previously pointed out in the Non-Final Rejection (mailed on 03/19/2008) and further clarified hereinafter, the Examiner respectfully submits that Te'eni clearly discloses "analyzing the sixth plurality of metadata to determine an eight plurality of potential conflicts between the second plurality of software components to be installed and the first plurality of software components previously installed on the system" *(see Column 1: 61-64, "When performing the predefined procedures necessary for an upgrade to be implemented frequently dependency conflicts may arise among the components present and the components to be installed."; Column 5: 10-20, "Virtual upgrade module 36 creates upgrade processes for the following tasks that are performed sequentially: (a) to collect all the information necessary for the dependency analysis and the dependency conflicts resolving*

process such as the relevant component data information units from component data table 28, the encoded dependency rules from xor-rules table 32 and from add-remove rules table 34 module, (b) to activate conflict resolver module 40 in order to check for potential dependency conflicts and to resolve any dependency conflicts that might arise as a result of the planned installation process ..."). Note that the virtual upgrade module collects and analyzes all the information necessary for the dependency analysis to check for potential dependency conflicts.

Third, Examiner further submits that the process of analyzing metadata to determine potential conflicts is well-known to one of ordinary skill in the computing art and also conventional in the area of software installation. By way of an example and not of limitation, software installation tools commonly include a software component dependency analyzer to collect and analyze software component dependency information and alert the user if a potential dependency conflict exists.

Fourth, with respect to the Applicant's contention that Claim 1 refers to a pre-deployment analysis used to prevent conflicts from occurring, the Examiner respectfully submits that the recitation of "a pre-deployment analysis of a plurality of software components of an application prior to deployment of the application" has not been given patentable weight because the recitation occurs in the preamble. A preamble is generally not accorded any patentable weight where it merely recites the purpose of a process or the intended use of a structure, and where the body of the claim does not depend on the preamble for completeness but, instead, the process steps or structural limitations are able to stand alone. See *In re Hirao*, 535 F.2d 67, 190 USPQ 15 (CCPA 1976) and *Kropa v. Robie*, 187 F.2d 150, 152, 88 USPQ 478, 481 (CCPA 1951).

Therefore, for at least the reasons set forth above, the rejection made under 35 U.S.C. § 103(a) with respect to Claim 1 is proper and therefore, maintained.

Conclusion

15. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

16. Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Qing Chen whose telephone number is 571-270-1071. The Examiner can normally be reached on Monday through Thursday from 7:30 AM to 4:00 PM. The Examiner can also be reached on alternate Fridays.

If attempts to reach the Examiner by telephone are unsuccessful, the Examiner's supervisor, Wei Zhen, can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Q. C./

Examiner, Art Unit 2191

/Wei Y Zhen/

Supervisory Patent Examiner, Art Unit 2191